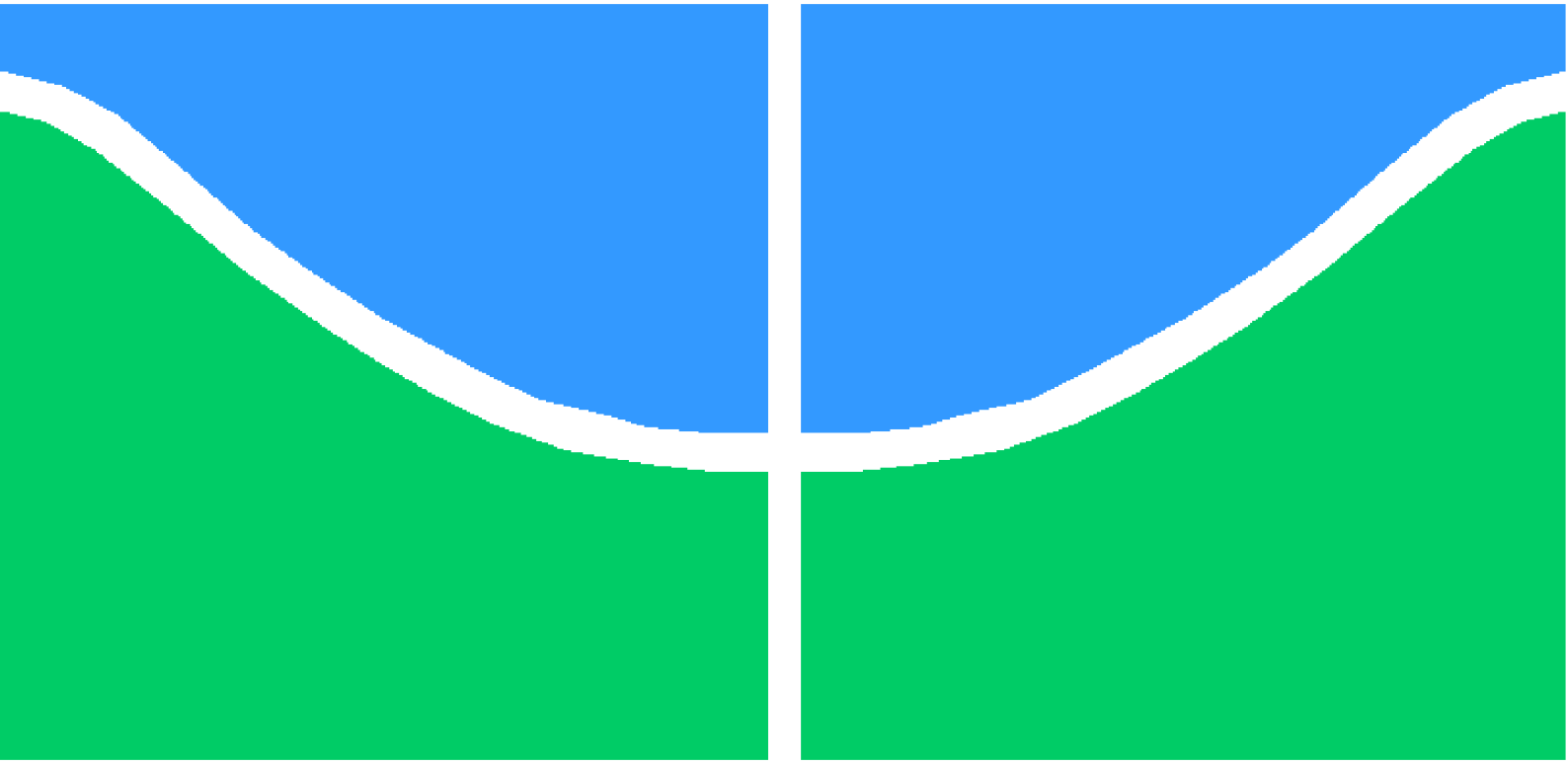

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Um Framework de MMORPG: Desenvolvimento Orientado Pela Arquitetura

Autor: Tales Martins de Souza
Orientador: Prof. Dr. Fabricio Ataides Braz

Brasília, DF
2013

Tales Martins de Souza

Um Framework de MMORPG: Desenvolvimento Orientado Pela Arquitetura

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Fabricio Ataides Braz

Brasília, DF

2013

Tales Martins de Souza
Um Framework de MMORPG: Desenvolvimento Orientado Pela Arquitetura/ Tales Martins de Souza. – Brasília, DF, 2013-
59 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Fabricio Ataides Braz

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2013.

1. MMORPG. 2. Framework. I. Prof. Dr. Fabricio Ataides Braz. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Um Framework de MMORPG: Desenvolvimento Orientado Pela Arquitetura

CDU 02:141:005.6

Tales Martins de Souza

Um Framework de MMORPG: Desenvolvimento Orientado Pela Arquitetura

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 10 de dezembro de 2013:

Prof. Dr. Fabricio Ataides Braz
Orientador

Profa. Dra. Carla Silva Rocha Aguiar
Convidado 1

Prof. Dr. Nilton Correia da Silva
Convidado 2

Brasília, DF
2013

Resumo

Os MMORPGs (Massively Multiplayer Online Role Playing Games) são jogos de computadores que suportam centenas ou milhares de jogadores simultaneamente que controlam um ou mais personagens, interagindo com o ambiente do jogo, completando missões, manipulando objetos do mundo virtual, conversando com outros personagens e combatendo monstros. Este gênero possui a sua jogabilidade como fator crítico para o sucesso, que é significativamente influenciada pela performance dos servidores. Os MMORPGs têm aumentado a sua expressão no mercado de jogos nos últimos anos, entretanto observa-se uma lacuna ferramental que favoreça o reúso no momento da implementação de jogos dessa natureza. Por meio do estudo dos problemas enfrentados no desenvolvimento de novos jogos MMORPGs e do levantamento de requisitos para este tipo de sistema, este trabalho de conclusão de curso apresenta uma proposta de um framework que implementa características essenciais de uma arquitetura de software voltada para MMORPG, com o intuito atenuar essa lacuna.

Palavras-chaves: MMORPG. framework. reúso.

Abstract

The MMORPGs (Massively Multiplayer Online Role Playing Games) are computer games that supports hundreds or thousands of concurrent players controlling one or more characters, interacting with the game environment, completing missions, manipulating objects in the virtual world, talking with other players and fighting monsters. This genre has the gameplay as a critical factor for the success, which is significantly influenced by the performance of the servers. MMORPGs have increased its expression in the gaming market in recent years, however, there is a gap tooling that promotes reuse at the time of implementation of such games. Through the study of the problems faced in developing new games of MMORPGs and gathering requirements for this type of system, this work presents a proposal for a framework that implements the essential features of a software architecture oriented to MMORPG in order to mitigate this gap.

Key-words: MMORPG. framework. reuse.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ADL	*Architecture Description Language*
APDU	*Application Protocol Data Unit*
BD	Banco de Dados
BOT	Robot
CN	Conector Arquitetural
CP	Componente Arquitetural
MMOG	*Massively Multiplayer Online Games*
MMORPG	*Massively Multiplayer Online Role Playing Game*
NPC	*Non-Player Character*
P2P	*Peer-to-Peer*
RF	Requisito Funcional
RNF	Requisito Não Funcional
RPG	Role-playing Game
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*

Sumário

1 Introdução

Este trabalho é uma monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software. O trabalho está organizado de forma a inicialmente contextualizar o leitor sobre o assunto tratado, os objetivos deste trabalho e a forma que o trabalho foi conduzido. Posteriormente é apresentado os resultados do trabalho.

1.1 Contextualização

Os MMORPGs (Massively Multiplayer Online Role Playing Games) são jogos de computadores onde os jogadores controlam um ou mais personagens que interagem com o ambiente do jogo, completando missões, manipulando objetos do mundo virtual, conversando com outros personagens e combatendo adversários. Estes jogos não possuem condições predefinidas de vitória ou derrota, grande parte da atração destes jogos está em termos de sua natureza como uma comunidade virtual, muitos jogadores passam horas conectados conversando com amigos, formando grupos e aperfeiçoando seu personagem com o tempo. Atualmente, o modelo econômico dos MMORPGs depende fortemente do esquema de inscrição dos jogadores, onde os jogadores pagam para jogar por um determinado tempo e, caso desejem jogar mais, pagam novamente (CORNETT, 2004).

Estimativas recentes mostram que nos últimos anos a receita do mercado de MMORPGs ultrapassou o valor de dois bilhões de dólares (KAWALE; PAL; SRIVASTAVA, 2009). Mercado este que é dominado pelo MMORPG World of Warcraft[1] que detém 62,4% das inscrições e atingiu a marca de dez milhões de jogadores inscritos em 2008 (WOODCOCKL, 2008).

Este gênero de jogo possui um modelo de negócio em que a jogabilidade é um fator crítico para o sucesso. A performance dos servidores influenciam significativamente na jogabilidade, logo é necessário priorizar essa característica durante o desenvolvimento do sistema (HUANG; YE; CHENG, 2004). Jogos online, especialmente os MMOG (Massive Multiplayer Online Games) se tornaram populares, tendo uma participação significativa no tráfego da internet. Cerca de 3 a 4% do tráfego da internet são atribuídos a seis jogos populares segundo Chen et al. (2005) na época de sua pesquisa. O tráfego dos jogos online devem crescer 37% no período de 2009 a 2014, o segundo maior crescimento de tráfego, atrás apenas do tráfego dos vídeos (SUZNJEVIC; STUPAR; MATIJASEVIC, 2011).

[1] World of Warcraft é um MMORPG desenvolvido pela Blizzard, 2004-2013, disponível em http://us.battle.net/wow/pt/ .

A maioria dos MMOGs comerciais são implementados como clientes e servidores, onde o estado global do jogo é gerenciado de maneira centralizada nos servidores. O controle centralizado apresenta significativas vantagens na segurança do sistema (YAMAMOTO et al., 2005). Devido ao grande custo para manter grandes servidores, foram feitos diversos trabalhos para buscar arquiteturas com um menor custo. Arquiteturas P2P (Peer-To-Peer) apresentaram um grande ganho de performance, já que o processamento do estado do jogo é feito nos clientes. Mas esta arquitetura apresenta um sério problema com trapaças, pois jogadores podem burlar as regras do jogo, uma vez que o processamento é feito em seu computador (KNUTSSON et al., 2004).

Segundo Bass, Clements e Kazman (2003) "a arquitetura de software de um programa ou sistema computacional é a estrutura ou estruturas do sistema que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles". A arquitetura é a representação que permite a análise da efetividade do projeto em satisfazer a seus requisitos declarados, onde alternativas arquiteturais são consideradas, enquanto alterações ainda são relativamente fáceis e busca reduzir os riscos associados à construção do software (PRESSMAN, 2006). Segundo Bass, Clements e Kazman (2003), a arquitetura de software é importante para:

- Facilitar a comunicação entre todas as partes interessadas no desenvolvimento do sistema.
- Destacar decisões iniciais de projeto que terão impacto profundo em todo o trabalho de engenharia de software que se segue, e no sucesso final do sistema como uma entidade operacional.
- Constituir um modelo relativamente pequeno, inteligível de como o sistema é estruturado e como seus componentes trabalham em conjunto.

Já os estilos e padrões são um conjunto de abstrações que podem ser aplicados a diversos sistemas de modo previsível, gerados por experiências em projetos similares (PRESSMAN, 2006). Padrões arquiteturais expressam um esquema estrutural da arquitetura, definindo subsistemas, especificando suas funcionalidades e incluindo regras e guias de relacionamento entre eles. Padrões de projeto expressam um esquema que refina os subsistemas, componentes ou relações entre componentes de um sistema de software de modo a resolver problemas de projetos recorrentes para determinado contexto (GAMMA et al., 1995 apud PRESSMAN, 2006).

Um framework, também chamado de arcabouço, é uma infraestrutura do esqueleto de implementação que fornece a estrutura e o compormentamento genérico para uma família de abstrações de software dentro de um contexto, colaborando para o desenvolvimento de novas aplicações daquele domínio (PRESSMAN, 2006). Diferentemente de bibliotecas,

os frameworks estão voltados para um domínio particular e possuiem um papel cada vez mais importante no desenvolvimento de software (FAYAD; SCHMIDT, 1997).

1.2 Problema

O problema identificado e atacado neste trabalho de conclusão de curso é a falta de padronização em sistemas que implementam MMORPGs.

Não há padronização ou mesmo arquiteturas de fato para sistemas de jogos online, diferentemente das aplicações web. Diferentes jogos costumam possuir configurações de hardware e software totalmente diferentes, forçando um investimento em hardware dedicado para cada jogo, mesmo se os recursos de outros jogos estiverem disponíveis (LEE; CHEN, 2010).

Sistemas que não reutilizam experiências de projetos passados, como decisões de projeto e códigos, apresentam uma menor produtividade em seu desenvolvimento e uma maior dificuldade de serem mantidos (RIEHLE, 2000).

Não existem regulamentações para o desenvolvimento, nem padronização dos requisitos de MMORPGs. Os desenvolvedores de novas aplicações de MMORPG precisam fazer um trabalho investigativo nas soluções existentes para definir os requisitos básicos de um MMORPG para aceitação no mercado.

1.3 Objetivo Geral

O objetivo geral deste trabalho de conclusão de curso é oferecer recursos que favorecam a padronização de sistemas que implementam MMORPG.

1.4 Objetivos Específicos

Para alcançar o objetivo geral, o trabalho foi dividido em duas partes. Os seguintes objetivos deverão ser alcançados na primeira parte:

- Consolidar o entendimento sobre as necessidades mínimas de um MMORPG.
- Definir a arquitetura de um framework que provê a solução das necessidades consolidadas.
- Definir as partes do framework que serão implementadas na segunda parte do trabalho.

E para a segunda parte do trabalho deverão ser alcançados o seguinte objetivo:

- Implementar a parte do framework definida na primeira parte do trabalho.

1.5 Justificativa

Conforme apresentado na seção 1.1, os MMORPGs representam uma importante e crescente parcela do mercado de jogos. Apesar disso, observa-se uma lacuna em soluções de reúso de código e decisões de projeto, devido a pouca padronização e oferta de soluções que favoreçam o reúso.

Este trabalho, então, busca suprir a falta de padronização por meio da construção de um framework, em especial um framework orientado a objetos pois os objetos são componentes potencialmente reusáveis por serem encapsulamentos independentes de estado e operações (SOMMERVILLE et al., 2008).

Frameworks de aplicações orientadas a objeto são uma tecnologia para prover decisões de projeto de software e implementações de maneira a aumentar a qualidade de software, por prover benefícios de modularidade, reusabilidade, extensibilidade e inversão de controle (FAYAD; SCHMIDT, 1997). Sistemas baseados em frameworks tendem ser mais fáceis de manter, costumam ter menos bugs por reusar códigos maduros e tendem a ser mais homogêneos (um conjunto de aplicações que utilizam um mesmo framework possuem arquitetura e implementação muito similares) (RIEHLE, 2000).

A modularidade é provida pelo uso de frameworks por encapsular detalhes de implementações instáveis por trás de interfaces estáveis, o que facilita a localização de impactos em mudanças de projeto e implementação (FAYAD; SCHMIDT, 1997). A modularidade é o atributo de software que permite a um programa ser gerenciável intelectualmente e é importante para viabilizar o planemento (PRESSMAN, 2006).

A reusabilidade, foco deste trabalho, é elevada pelos frameworks por apresentarem interfaces estáveis e definições de componentes genéricos que são utilizados em novas aplicações. A reusabilidade evita que soluções recorrentes do domínio da aplicação sejam recriadas e revalidadas (FAYAD; SCHMIDT, 1997). Segundo a ISO/IEC (2001) a reusabilidade é definida como "o grau em que um módulo de software ou outro produto pode ser utilizado em mais de um programa de computador ou sistema de software".

A extensibilidade é provida pelas métodos gancho dos frameworks. Os métodos gancho são métodos que permitem que os desenvolvedores insiram variações no comportamento da aplicação para estender as interfaces e comportamentos padrão do framework (FAYAD; SCHMIDT, 1997). A extensibilidade é a maneira como novas funcionalidades vindas de demandas ou novas tecnologias podem ser desenvolvidas e exploradas pelo sistema sem perder suas capacidades (HENTTONEN MATINLASSI; KANSTéN, 2007).

A inversão de controle é uma característica muito comum da arquitetura em tempo

de execução de vários frameworks. A chamada aos métodos gancho não é feita diretamente pelo programador, a chamada é feita pelo framework, pois este é responsável por fazer a manipulação de eventos (FAYAD; SCHMIDT, 1997).

Para que este framework seja implementado corretamente foram definidos os objetivos específicos da primeira parte deste trabalho na seção 1.4.

A consolidação do entendimento sobre as necessidades mínimas de um MMORPG é necessária, pois não há normas que regulamentam MMORPGs. Como o framework precisa atender as necessidades genéricas dos MMORPGs, é necessário uma pesquisa das características comuns nos MMORPGs existentes, caso contrário o domínio não será bem representado pelo framework.

A definição da arquitetura é de crítica importância para o atendimento a vários atributos de qualidade de interesse em um sistema e ,também, permite que os atributos de qualidade sejam avaliados ainda em nível de arquitetura, ou seja poupa esforços por realizar a avaliação antes da implementação do sistema (BASS; CLEMENTS; KAZMAN, 2003).

A definição das partes do framework a serem implementadas é importante para a segunda parte do trabalho, que irá implementar parte do framework proposto. Devido a limitações de tempo e quantidade de desenvolvedores não será possível a implementação de todo o framework. O restante do framework poderá ser implementado em trabalhos futuros.

A segunda parte do trabalho que consiste na implementação e teste da parte do framework definida é importante para que o framework possa ser utilizado e assim atender o objetivo geral do trabalho. Os testes são necessários para evidenciar objetivamente que as necessidades consolidadas foram atendidas.

1.6 Metodologia

O desenvolvimento deste trabalho foi dividido em cinco etapas. As quatro primeiras etapas foram definidas para a primeira parte do trabalho e a quinta etapa foi definida para a segunda parte do trabalho.

1.6.1 Etapa 1

A primeira etapa deste trabalho foi a delimitação e reconhecimento de domínio de MMORPGs. Esta etapa identifica e caracteriza os principais problemas técnicos no desenvolvimento de MMORPGs atualmente. A etapa 1 também identifica as soluções adotadas para estes problemas.

Por meio de pesquisa bibliográfica foi identificado áreas de estudo que têm influência para as decisões de projeto de um sistema de MMORPG. Esta etapa foi organizada de acordo com estas áreas de estudo.

As palavras-chaves que conduziram este estudo foram: mmorpg, mmog, *computer architecture*, *traffic analysis*, *network servers*, *resource allocation*, *game server performance*, *load balalancing*, *peer-to-peer*, *memory management*, *scalability*, *TCP performance*, *modeling* e *real-time systems*.

1.6.2 Etapa 2

A etapa 2 revela as necessidades do sistema MMORPG por meio da definição dos requisitos. Esta definição permite um melhor entendimento dos objetivos que o framework deverá alcançar. O levantamento de requisitos foi feito com base na análise de alguns MMORPGs bem sucedidos atualmente.

1.6.3 Etapa 3

Esta etapa identificou as principais decisões de projeto que o framework seguirá, de forma a solucionar os problemas identificados na etapa 1 e atender os requisitos levantados na etapa 2.

Para representar a arquitetura foi utilizada a notação xADL, que é uma linguagem para descrição de arquitetura (ADL) desenvolvida pelo *Institute for Software Research* na *University of California, Irvine*. A arquitetura é descrita por meio do uso de Componentes (CPs), interfaces, Conectores (CNs) e configurações. A notação xADL é extensível, permitindo incluir modificações para representar melhor a arquitetura (KHARE et al., 2001).

Componentes são unidades de software independentes que podem ser compostas por outros componentes para criar um sistema de software. Os componentes são definidos por suas interfaces, que definem os serviços fornecidos pelo componente e quais serviços devem ser fornecidos pelos outros componentes (SOMMERVILLE et al., 2008).

Para sistemas de grande escala, a implementação da interação entre os componentes são muitas vezes mais desafiadoras do que a implementação dos componentes. Um conector é uma entidade que gerencia a interação entre componentes. A complexidade dos conectores podem variar significativamente (TAYLOR; MEDVIDOVIC; DASHOFY, 2009)

As configurações ou topologias são um conjunto específico de associações entre os componentes e conectores que descrevem a aquitetura de um software (BASS; CLEMENTS; KAZMAN, 2003).

1.6.4 Etapa 4

A etapa 4 consiste na seleção dos elementos da arquitetura que serão implementados na segunda parte do trabalho.

Os componentes e conectores foram mapeados nos requisitos funcionais e não funcionais. E com a definição de critérios, os requisitos foram priorizados; permitindo que os elementos da arquitetura fossem selecionados.

1.6.5 Etapa 5

A última etapa do trabalho consiste na implementação da parte alvo do framework. Esta etapa analisa a degradação da arquitetura pela comparação da arquitetura descritiva apresentada na etapa 3 e a arquitetura implementada. A etapa 5 consiste também na realização de testes para verificar o atendimento aos requisitos levantados e selecionados.

1.7 Estrutura do Trabalho

O capítulo 1 apresentou a contextualização do trabalho, os seus objetivos e a forma que foi conduzido. O capítulo 2 apresenta os resultados obtidos nas etapas definidas na metodologia. E por fim, o capítulo 3 apresenta as considerações finais do trabalho.

2 Execução

Este capítulo contém a apresentação dos resultados das etapas definidas na seção 1.6. A primeira etapa delimita e reconhece o domínio de MMORPGs. A segunda etapa define as necessidades em comuns dos MMORPGs. A terceira etapa define as principais decisões de projeto do framework. A quarta etapa seleciona o escopo do framework a ser desenvolvido na segunda parte do trabalho. E a quinta etapa aborda o desenvolvimento da parte alvo do framework e a degradação da arquitetura em relação a arquitetura descritiva definida na terceira etapa do trabalho.

2.1 Resultados da Etapa 1

Conforme apresentado na seção 1.6.1, a Etapa 1 consistiu na delimitação e reconhecimento de domínio de MMORPGs.

O estudo bibliográfico no assunto identificou três áreas de interesse para este trabalho:

- Modelagem: área que busca propor algoritmos, organização de servidores, camadas e módulos de software para a implementação de sistemas de MMORPG.
- Análise de tráfego: área que estuda o tráfego de pacotes entre clientes e servidores de MMORPGs, buscando a identificação de situações de maiores congestionamentos para evidenciar oportunidades de melhoria de desempenho.
- Detecção de clientes automáticos: é composto de técnicas ou algoritmos para identificar jogadores controlados por software e não um jogador humano.

2.1.1 Modelagem

MMOGs normalmente possuem um processamento centralizado que é capaz de prover grande segurança e facilidade de implementação, mas é possível utilizar uma arquitetura *Peer-to-Peer* (P2P) em que os "nós" de jogadores realizam o processamento de uma região do jogo, reduzindo o custo de manutenção e permitindo um número maior de jogadores no mesmo mundo virtual (DIOT; GAUTIER, 1999).

O método P2P para troca de mensagens entre os jogadores, proposto por Diot e Gautier (1999), permite que áreas sejam distribuídas aos jogadores de modo que cada jogador nunca processe acima de um número máximo de jogadores. O método também apresenta um sistema de backup de nós, para evitar perdas com falhas no nó que esteja

processando uma área. Todo o espaço do jogo é dividido em áreas iguais que são alocadas a jogadores, que terão que receber, processar e responder a todos os jogadores dentro da área. Os jogadores responsáveis pelas áreas são eleitos por capacidade de processamento e disponibilidade de banda por meio de um servidor centralizado, onde ocorre também a autenticação dos jogadores.

Embora a arquitetura P2P apresente muitas vantagens, Knutsson et al. (2004) identificou alguns problemas que inviabilizam esta arquitetura:

- O jogo tem atualizações frequentes que precisam ser propagados com certa limitação de tempo, *peers* costumam ter redes limitadas já que estão nos limites da rede.
- Replicação dos estados para aumentar disponibilidade apresenta o problema de controle de estados inválidos já que um jogador pode ficar offline sem notificar outros *peers*.
- Manter um grande número de réplicas causam problemas de performance.
- A prevenção de roubo de contas e a prevenção de fraudes são complicadas pois o jogo será processado nos peers, que podem alterar o fluxo de dados original.

A implementação de um MMORPG utilizando o protocolo *Transmission Control Protocol* (TCP) não é mais lenta que uma utilizando *User Datagram Protocol* (UDP) pois o *buffer* de envio normalmente está vazio, já que um evento deve ser enviado imediatamente (GRIWODZ; HALVORSEN, 2006).

Uma arquitetura *proxy* pode economizar até 40% dos recursos no servidor. Multiplexar os fluxos de eventos para os servidores *proxy* reduz o processamento de geração dos cabeçalhos nos pacotes TCP e reduz também a quantidade de eventos enviados, já que vários clientes podem compartilhar um mesmo evento (GRIWODZ; HALVORSEN, 2006).

A maioria dos MMORPGs utilizam uma arquitetura *"sharded"* que provê múltiplos mundos virtuais idênticos para aumentar a escalabilidade de jogadores, mas essa arquitetura também sub-utiliza servidores que acabam ficando com mundos desertos. Como as interações do jogador com o jogo ocorrem principalmente na região em que se situa, é possível particionar o processamento do jogo. Tal particionamento apresentou resultados experimentais de economia de 52% no número de servidores e uma redução de 62% no consumo de energia, enquanto a experiencia de jogo não foi alterada (LEE; CHEN, 2010).

2.1.2 Análise de tráfego

Um aspeto importante para a qualidade de serviço é a rede. Para o jogo ser imersivo, o mundo virtual precisa ser responsivo, ou seja, a rede precisa de um baixo tempo

de atraso (SUZNJEVIC; STUPAR; MATIJASEVIC, 2011).

Resultados experimentais apresentados por Griwodz e Halvorsen (2006) mostraram que apesar dos fluxos de eventos entre cliente e servidor serem pequenos, o servidor apresenta uma quantidade alta de fluxos, que juntas acabam congestionando a rede.

A modelagem do tráfego em um conjunto de categorias de ações que representam uma situação no mundo virtual, feita por Suznjevic, Stupar e Matijasevic (2011), mostra que:

- A atividade de realização de missões é a atividade mais geral do jogo, com a maior variância de tipos de ações e é a atividade predominante para novos jogadores.
- Batalhas jogador contra jogador é a categoria mais dinâmica em termos de entradas do usuário, que resulta na maior taxa de transmissão de pacotes para o tráfego no cliente e também é a atividade com tamanhos mais distintos de *Application Protocol Data Unit*s (APDUs).
- A exploração de cavernas com grupos pequenos é a categoria "média", em que as características de tráfego estão entre as atividades de jogadores sozinhos e as atividades de grandes grupos.
- A atividade de exploração de grandes cavernas com grupos grandes é a atividade que mais gera informação para o servidor, pois o tráfego carrega uma carga de transmisão de dados maior já que haverá um grande número de jogadores e personagens controlados pelo sistema dentro da área de interesse de cada jogador, e suas atualizações precisam ser informadas a todos os jogadores envolvidos.

A análise das conexões do Anarch Online[1] evidencia que o maior desafio para este tipo de tráfego não é a largura de banda e sim o tempo de espera das retransmissões, pois poucos pacotes são enviados por segundo, deixando a janela de congestionamento pequena e a perda de pacotes é detectada apenas por término de um contador de tempo. Uma maneira melhorar o desempenho dos servidores é modificar o kernel do sistema operacional para se comportar de maneira mais agressiva nos *timeouts* de retransmissões.

2.1.3 Detecção de clientes automáticos

Clientes que jogam automaticamente, também chamados de *Robot*s (BOTs), são programas possivelmente com inteligência artificial para gerar benefícios a um jogador, como juntar dinheiro ou itens por derrotar monstros ou aumentar a perícia em uma habilidade como pesca. Este ganho sem o devido esforço acaba desbalanceando o jogo, fazendo com que vários jogadores deixem o jogo (THAWONMAS; KASHIFUJI; CHEN,

[1] Anarch Online é um MMORPG de ficção científica produzido pela Funcon em 2001.

2008). Um dos maiores desafios enfrentados pelos MMORPGs de sucesso é o uso de clientes que jogam automaticamente (CHEN et al., 2006).

Pelo fato dos BOTs não desrespeitarem nenhuma regra lógica do jogo sua detecção é difícil. Muitas vezes a detecção é feita por funcionários, que patrulham as zonas do jogo e questionam jogadores suspeitos. Mas esta prática não é eficiente e o uso de técnicas automáticas para a detecção de BOTs são recomendadas (CHEN et al., 2006).

Uma técnica de detecção de BOTs é pela identificação de discrepâncias nos padrões de tráfego entre clientes e servidor.

Um dos fatores mais intuitivos de discrepância entre BOTs e jogadores humanos é também um dos mais relevantes, o tempo de reação a um dado evento. Como jogadores humanos reagem pressionando uma tecla do teclado ou clicando em algo pelo mouse, sua reação é bem mais lenta que a reação de um BOTs que toma a ação por meio da execução de um trecho de código. Mas existem BOTs que apresentam um tratamento sofisticado para o tempo de resposta, que apresenta resultados similares ao tempo de resposta de um jogador humano (THAWONMAS; KASHIFUJI; CHEN, 2008).

Outro padrão de tráfego a ser analisado é a regularidade no tempo de resposta. Não é observável padrão na regularidade tempo de resposta dos jogadores humanos, enquanto é observável padrões para BOTs (CHEN et al., 2006).

O volume de tráfego também apresenta padrões e podem ser analisados. Os BOTs apresentam um volume de tráfego com variações mais suaves do que jogadores humanos (CHEN et al., 2006).

A sensibilidade às condições da rede é o padrão a ser analisado que apresenta melhores resultados. Foi observado que os jogadores humanos se adaptam naturalmente a lentidão na rede, assim sua taxa de envio de pacotes acompanha as mudanças na taxa de recebimento de pacotes. Mas o mesmo não acontece para os BOTs que possuem um *loop* de execução independente do tempo de recebimentos dos pacotes, caso um pacote atrase, mais ações são acumuladas para serem realizadas (CHEN et al., 2006).

Considerando estes padrões Chen et al. (2006) propôs um esquema conservativo que apresentou 0 falsos positivos (julgar um jogador humano como BOTs) e 90% de decisões corretas com 10000 pacotes analisados. E um esquema mais agressivo que apresentou menos de 1% de falsos positivos e 95% de decisões corretas com um 2000 pacotes analisados.

2.2 Resultados da Etapa 2

Conforme apresentado na seção 1.6.2, a Etapa 2 consistiu na definição das necessidades de um sistema de MMORPG. Esta definição é importante para que o framework

possa contemplar as necessidades comuns e atenda a expectativa da maioria dos desenvolvedores de MMORPGs. A definição dessas necessidades foram feitas como requisitos funcionais e requisitos não funcionais.

A identificação dos requisitos foi feita por meio de estudo bibliográfico e análise dos seguintes MMORPGs de sucesso: "World of Warcraft", "Ragnarök Online[2]", "Cabal Online[3]", "Rising Force Online[4]", "Star Wars: The Old Republic[5]" e "TERA: The Exiled Realm of Arborea[6]".

2.2.1 Requisitos Funcionais

Segundo Sommerville et al. (2008) um Requisito Funcional (RF) é a declaração de serviços que o sistema deve ou não deve fornecer, como o sistema deve reagir a entradas específicas ou como o sistema deve se comportar em determinadas situações. Os requisitos funcionais identificados foram:

- RF1: O sistema deve manter todos os jogadores com softwares clientes atualizados.
- RF2: O sistema deve permitir a criação, atualização e deleção de contas de usuários.
- RF3: O sistema deve autenticar os jogadores para entrada no mundo virtual.
- RF4: O sistema deve permitir a criação personalizada e deleção dos avatares dos jogadores.
- RF5: O sistema deve manter o estado do avatar para quando o seu respectivo usuário retornar ao sistema.
- RF6: O sistema deve manter um mundo virtual com avatares, *Non-Player Characters* (NPCs), objetos de interação(como cadeiras e caixas de correio) e objetos decorativos.
- RF7: O sistema deve permitir que os jogadores visualizem os acontecimentos no mundo virtual de modo sincronizado.
- RF8: O sistema deve receber comandos do jogador para movimentação e interação do avatar com o mundo virtual.

[2] Ragnarök Online é um MMORPG desenvolvido pela Gravity Corp., 2002-2013, disponível em http://www.ragnarokonline.com/ .

[3] Cabal Online é um MMORPG desenvolvido pela ESTsoft, 2008-2013, disponível em http://cabal.gamemaxx.com.br .

[4] Rising Force Online é um MMORPG desenvolvido pela CCR Inc., 2004-2013, disponível em http://rfonline.gamescampus.com/ .

[5] Star Wars: The Old Republic é um MMORPG desenvolvido pela BioWare em parceria com a LucasArts., 2011-2013, disponível em http://www.swtor.com/ .

[6] TERA: The Exiled Realm of Arborea é um MMORPG desenvolvido pela Bluehole Studios, 2011-2013, disponível em http://tera.enmasse.com/ .

- RF9: O sistema deve permitir que os jogadores se agrupem em grupos (2 a 5 jogadores), raides (5 a 50 jogadores) ou guildas (mais de 10 jogadores).
- RF10: O sistema deve permitir a comunicação dos jogadores diretamente entre si e por regiões, grupos, guildas ou raides.
- RF11: O sistema deve permitir o combate de jogadores contra NPCs e jogadores contra jogadores.
- RF12: O sistema deve oferecer seguintes elementos comuns em jogos de *Role-playing Game* (RPG): sistema de níveis, classes, itens, missões, profissões e habilidades.
- RF13: O sistema deve permitir a troca de itens entre os jogadores.
- RF14: O sistema deve desconectar um jogador caso a conexão com o servidor esteja inoperante (*keepalive*).
- RF15: O sistema deve permitir que jogadores reportem bugs e trapaças.

2.2.2 Requisitos Não Funcionais

Sommerville et al. (2008) também afirma que um Requisito Não Funcional (RNF) é uma restrição sobre os serviços ou funções oferecidos pelo sistema, incluindo restrições de *timming*, de processo de desenvolvimento e padrões, geralmente se aplicando ao sistema como um todo e não as suas características individuais. Os requisitos não funcionais foram identificados foram:

- RNF1: O sistema deve suportar pelo menos 1000 jogadores simultâneos por reino.
- RNF2: O sistema deve suportar pelo menos 100 jogadores em uma área equivalente ao campo de visão de um jogador.
- RNF3: O sistema deve possuir um atraso entre cliente e servidor inferior a 200 ms(DIOT; GAUTIER, 1999).
- RNF4: O sistema deve transmitir as alterações de posição e ações de combate com prioridade em relação as demais ações.
- RNF5: O sistema não deve permitir que um jogador trapaceie as regras do jogo (*exploits*).
- RNF6: O sistema deve suportar que o cliente se atrase até 5000 ms na comunicação com o servidor.

- RNF7: O sistema não deve permitir que a diferença da posição de um avatar mantida pelo servidor e pelo cliente seja maior que o diametro do avatar(DIOT; GAUTIER, 1999).
- RNF8: O sistema não deve permitir que falhas inesperadas como desconexão ou falha de sincronização em um cliente afetem os demais clientes (DIOT; GAUTIER, 1999).
- RNF9: O sistema deve manter uma comunicação segura entre cliente e servidor (criptografia).
- RNF10: O sistema deve permitir recuperação dos dados dos jogadores, itens e NPCs.
- RNF11: O sistema deve suportar a expansão de itens, habilidades, mundo virtual e número de jogadores.
- RNF12: O sistema deve possuir um armazenamento escalável de modo a não impor restrições no número máximo de itens, jogadores e NPCs.

2.3 Resultados da Etapa 3

Conforme apresentado na seção 1.6.3, a Etapa 3 consistiu na definição das principais decisões de projeto do framework.

A arquitetura foi descrita utilizando a linguagem xADL. Para facilitar a compreensão dos modelos, as cores dos componentes apresentados pelos diagramas foram modificadas de acordo com a sua natureza, sendo que os componentes de servidor ficaram em azul e os componentes do cliente ficaram em verde. Os conectores estão representadas pelos blocos de cor amarela.

A visão geral de um MMORPG é apresentado na Fig. (1).

Os MMORPGs apresentam um modelo econômico de inscrição, conforme apresentado na seção 1.1 que é muito dependente de uma presença online. Um portal web se faz necessário para que os jogadores obtenham mais informações sobre o jogo, criem suas contas, façam pagamentos e obtenham o jogo.

Para que o requisito funcional RF1 seja atendido, é necessário um controle na versão cliente que garanta a atualização para a sua última versão disponível. Tal responsabilidade é assumida pelo componente atualizador de cliente.

O portal web (representado pelo Web Browser e Website na Fig. (1)) e a atualização do cliente (representado pelo Atualizador do Cliente e Provedor de Atualizações de Arquivos na Fig. (1)) são componentes que dão suporte ao típico serviço do jogo MMORPG.

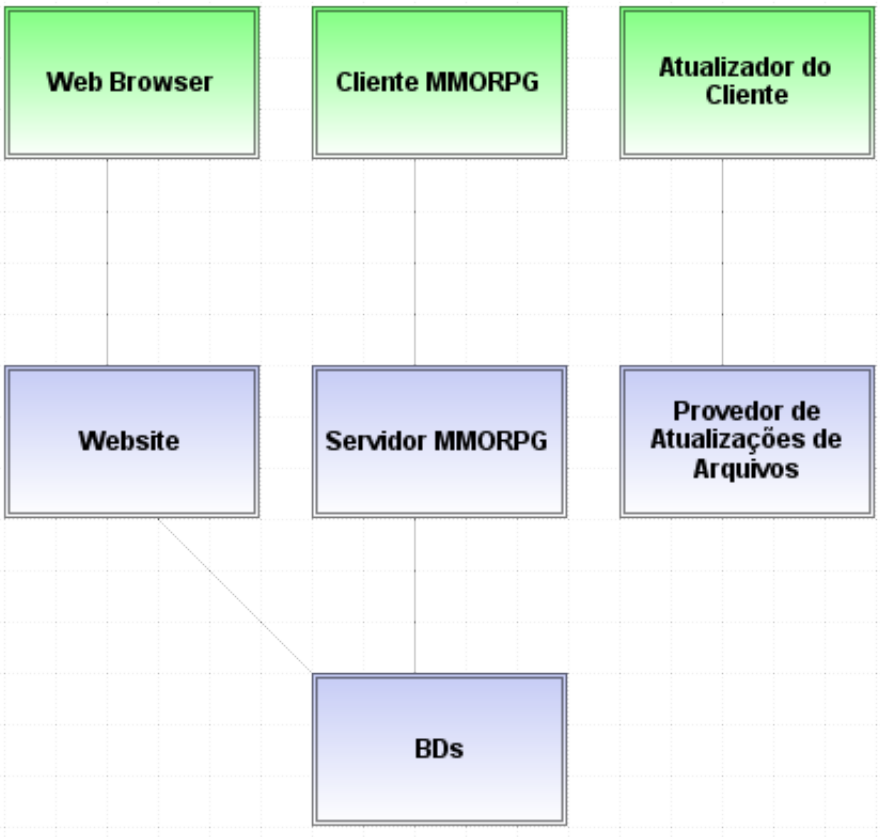

Figura 1 – Visão Geral da Arquitetura de um MMORPG

Atualmente existem diversos frameworks como Django[7], Ruby on Rails[8], Struts[9], Hibernate[10] e Catalyst[11] que dão suporte ao desenvolvimento do portal web e, por isso, serão desprezados pelo restante do trabalho.

O atualizador ou *downloader* também não será abordado no framework por ser um componente auxiliar para a existência do MMORPG. A implementação do atualizador pode variar entre as várias arquiteturas existentes, como a tradicional cliente-servidor ou *peer-to-peer*.

A arquitetura apresentada a seguir foi gerada com base em estudos bibliográficos, análise do funcionamento do MMORPG de maior sucesso, World of Warcraft, e análise do MMORPG de código livre Planeshift[12]

[7] Django é um framework para desenvolvimento rápido para web publicado sob a licença BSD em 2005.
[8] Ruby on Rails é um framework livre para aumentar a velocidade e facilidade no desenvolvimento de sites orientados a banco de dados.
[9] Struts é um framework de desenvolvimento da camada controladora de aplicações web, desenvolvido por Craig McClanahan e doado para a Apache Software Foundation em 2002.
[10] Hibernate é um framework para mapeamento objeto-relacional distribuido com a licença LGPL em 2001.
[11] Catalyst é um framework livre para desenvolvimento de aplicações web escritas em Perl que teve sua primeira release em 2005.
[12] Planeshift é um MMORPG desenvolvido pela Atonic Blue, 2000-2013, disponível em http://planeshift.it/ .

2.3.1 Cliente

Os componentes que operam no cliente não devem assumir a responsabilidade de processar as regras do jogo, pois isso favorece jogadores mal intencionados a burlarem tais regras, uma vez que eles podem comprometer o fluxo de execução autêntico do cliente, em benefício próprio (KNUTSSON et al., 2004).

A arquitetura do cliente proposta é responsável apenas por apresentar informações vindas do servidor e enviar os comandos feitos pelo jogador para ser processada no servidor. A comunicação entre cliente e servidor é otimizada para induzir a impressão do jogador de que a execução aconteça em tempo real.

A arquitetura lógica do cliente é apresentado na Fig. (2).

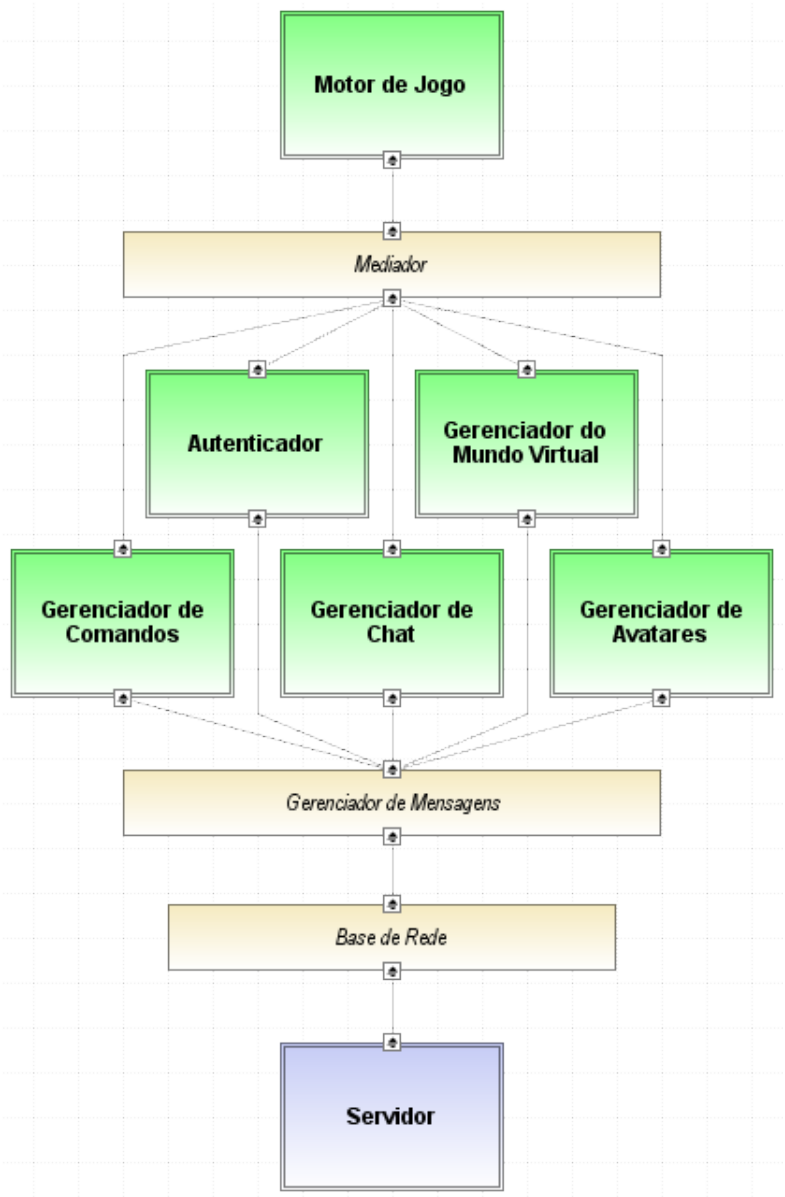

Figura 2 – Arquitetura Lógica do lado cliente do Framework

As funcionalidades dos componentes são descritas abaixo:

- CP1: O Motor de Jogo é um conjunto de bibliotecas para simplificar e abstrair o desenvolvimento de jogos. É responsável por renderizar gráficos 2D ou 3D, simular física, calcular animações, emitir sons, gerenciar memória e arquivos. Os motores de jogos também fornecem abstração de hardware, permitindo que os programadores desenvolvam sem a necessidade de conhecer a arquitetura da plataforma alvo. O framework deverá utilizar um motor de jogo popular como Unreal Development Kit[13], CryEngine[14] ou Torque Game Engine[15] para poder se focar nos aspectos de MMORPG.

- CP2: O Autenticador é o módulo responsável por autenticar o jogador no servidor. O Autenticador deverá trocar mensagens com o componente Servidor de Login do servidor para permitir a identificação do jogador. Como um mesmo software cliente pode ser utilizado por qualquer jogador, a identificação é de grande importância para que os avatares sejam carregados corretamente.

- CP3: O Gerenciador do Mundo Virtual é a parte do sistema responsável por manter o estado dos objetos visíveis no mundo virtual. O Gerenciador do Mundo Virtual deve atualizar o estado de todos os objetos pertencentes ao mundo virtual sempre que notificado pelo servidor.

- CP4: O Gerenciador de Comandos é o componente do framework responsável por comunicar ao servidor quais comandos o jogador está realizando, para poder ser validado e então transmitido para todos os jogadores no campo de visão. O componente também deve realizar uma validação prévia para economizar recursos do servidor e otimizar o tempo de resposta para o jogador.

- CP5: O Gerenciador de Chat é o componente do sistema responsável por gerenciar as mensagens de chat do sistema. O sistema deve ser capaz de enviar e receber mensagens em vários canais, como área, guilda, *raid*, grupo e comunicação direta entre jogadores.

- CP6: O Gerenciador de Avatares é a parte do cliente que mantém a lista de avatares do jogador. Este módulo deve receber a lista de avatares vinda do servidor e também deverá coordenar a criação de avatares, realizando tarefas de validação de nome e customizações do avatar.

Os conectores são descritos abaixo:

[13] Unreal Development Kit é um motor de jogo escrito em C++ e desenvolvido pela Epic Games
[14] CryEngine é um motor de jogo desenvolvido pela Crytek
[15] Torque Game Engine é um motor de jogo proprietário multiplataforma da GarageGames

- CN1: O Mediador é o conector responsável por intermediar a interação entre o motor de jogo os diversos componentes do cliente. O mediador deverá mapear as interações dos demais componentes em correspondentes no motor do jogo. Desta forma o utilizador do framework poderá utilizar o motor de jogo de seu interesse, tendo apenas que estender esse conector para realizar o mapeamento corretamente.

- CN2: O Gerenciador de Mensagens é a parte do cliente responsável por distribuir as mensagens vindas do servidor para os módulos que realizam sua interpretação. O Gerenciador de Mensagens deverá implementar o padrão de projeto *Publisher-Subscriber*, pois módulos diferentes podem estar interessados em uma mesma mensagem. Assim, módulos diferentes podem se inscrever dinamicamente para receber notificações de um mesmo tipo de mensagem (BUSCHMANN et al., 1996).

- CN3: A Base de Rede é o módulo do cliente que deverá transmitir e receber dados do servidor por meio da utilização da rede. A Base de Rede é responsável por fazer o controle de pacotes para otimizar a velocidade de conexão.

2.3.2 Servidor

A parte servidor do framework é projetada para ser executada em um conjunto de computadores para possibilitar a escalabilidade (horizontal e vertical). O servidor é responsável por processar todas as regras do jogo. A arquitetura lógica do servidor é apresentada na Fig. (3).

As funcionalidades dos componentes são descritas abaixo:

- CP7: O Servidor de Chat é o componente do servidor responsável por receber todas as mensagens de conversa dos jogadores e NPCs de um canal e transmiti-las para todos os jogadores daquele canal. O Servidor de Chat deverá oferecer os canais de grupo, guilda, mensagens privadas, área, busca por grupo, comércio e um canal especial que o jogador só verá as mensagens dos jogadores que estiverem em seu campo de visão.

- CP8: O Servidor de Criação de Avatares é responsável por criar os avatares dos jogadores. O módulo deverá permitir a customização do avatar e não deverá permitir que dois avatares existam com o mesmo nome.

- CP9: O Servidor de Login é a parte do servidor destinada a controlar o acesso do jogador. O Servidor de Login deverá autenticar o usuário, enviar sua lista de personagens e informá-lo qual região irá comunicar.

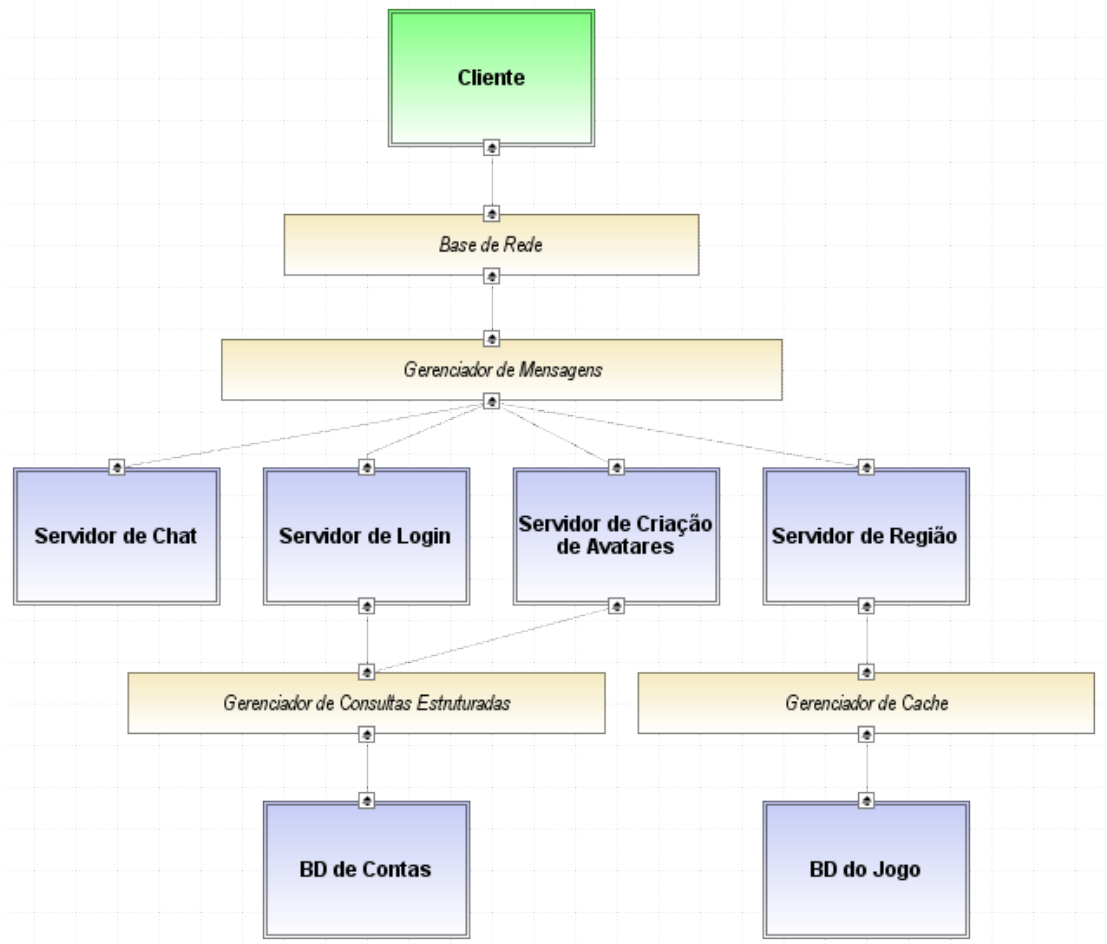

Figura 3 – Arquitetura Lógica do lado servidor do Framework

- CP10: O Servidor de Região é o componente responsável por realizar todo o processamento dos acontecimentos de uma parte do mundo virtual. A seção 2.3.2.1 se dedica a detalhar este componente.

- CP11: O Banco de Dados (BD) de Contas é o repositório que armazena todas as informações da conta usuário.

- CP12: O BD do Jogo é o repositório responsável por armazenar as informações dos avatares, habilidades, NPCs e itens. Este banco de dados é necessário para permitir o balanceamento de itens e habilidades no jogo.

Os conectores são descritos abaixo:

- CN4: A Base de Rede é a parte do servidor responsável por enviar e receber mensagens para os clientes. A Base de Rede é responsável por otimizar a utilização da rede, fazendo o controle do empacotamento das mensagens. Este conector pode utilizar servidores *proxy* para melhorar o desempenho (GRIWODZ; HALVORSEN, 2006).

- CN5: O Gerenciador de Mensagens é o conector responsável por distribuir as mensagens recebidas dos clientes para os módulos interessados. Assim como o conector Gerenciador de Mensagens do cliente, o Gerenciador de Mensagens do servidor deverá implementar o padrão de projeto *Publisher-Subscriber* para manter os componentes interessados atualizados (BUSCHMANN et al., 1996).

- CN6: O Gerenciador de Cache é a parte do sistema responsável por otimizar o acesso ao BD do Jogo. Para aumentar a performance do servidor, as lentas operações de escrita e leitura no banco de dados são substituídas por acesso a memória RAM (DEWITT; GRAY, 1992).

- CN7: O Gerenciador de Consultas Estruturadas é o conector responsável por facilitar o acesso ao BD de Contas. O Gerenciador de Consultas Estruturadas deverá abstrair o acesso ao banco de dados para facilitar a utilização pelos componentes Servidor de Login e Servidor de Criação de Avatares. O conector irá abstrair as chamadas *query* ao banco de dados relacional a interface orientada à objeto do conector.

2.3.2.1 Servidor de Região

O componente Servidor de Região é mais complexo que os demais componentes e está detalhado na estrutura apresentada na Fig. (4).

As funcionalidades dos componentes são descritas abaixo:

- CP13: O Verificador de Trapaças é o componente do sistema capaz de receber denúncias de trapaças e realizar a análise dos avatares denunciados. O Verificador de Trapaças deverá combinar as técnicas de detecção por comportamento, proposto por Thawonmas, Kashifuji e Chen (2008), e técnicas de detecção por tráfego, proposto por Chen et al. (2006).

- CP14: O Gerenciador de Profissões é responsável por gerenciar os pontos de profissão dos jogadores e a construção de itens por profissões. O componente deve verificar se o avatar possui pontos suficientes e os itens necessários para a criação do item requerido e também deverá apagar os itens consumidos durante o processo.

- CP15: O Gerenciador de Grupos é a parte do sistema responsável por manter os conjuntos de jogadores. O Gerenciador de Grupos deve permitir a criação de conjuntos do tipo guilda, raide ou grupo. Os conjuntos de raide ou grupo deverão possibilitar a configuração da distribuição de novos itens, que poderá ser cada um por si, dividido igualmente ou um membro é eleito para realizar a distribuição.

- CP16: O Gerenciador de Missões é o componente do sistema que gerencia as missões dos avatares. O Gerenciador de Missões deve permitir que o jogador aceite ou

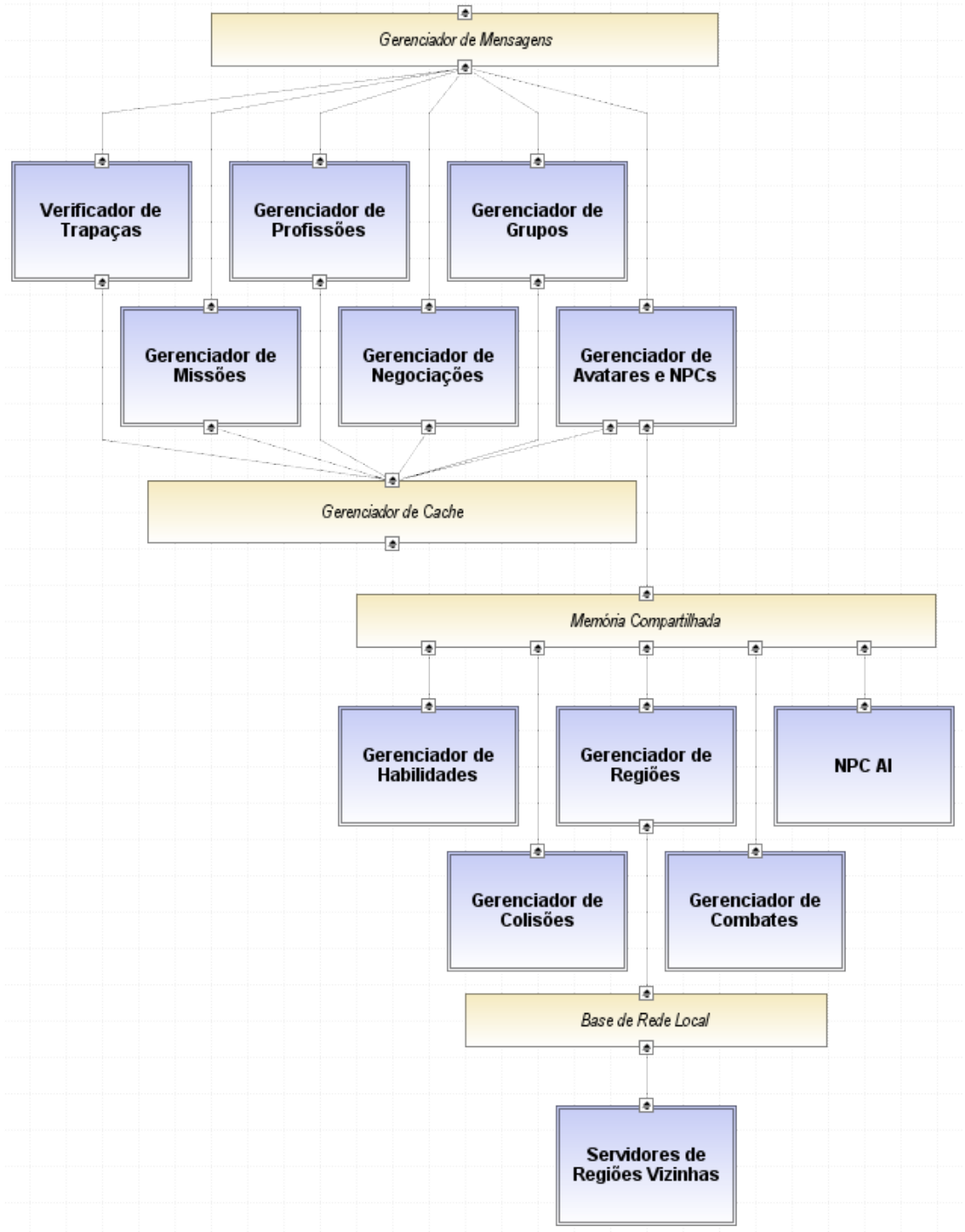

Figura 4 – Arquitetura lógica do componente Servidor de Região

abandone uma missão e também deverá controlar o progresso das missões aceitas. O componente deverá permitir que o avatar aceite a missão somente se possuir os pré-requisitos para aquela missão.

- CP17: O Gerenciador de Negociações é responsável pela troca de itens entre os jogadores. Cada jogador deverá confirmar sua proposta antes do componente efetivar a troca de itens. Caso a proposta seja alterada depois que um jogador já estiver

confirmado, este deverá confirmar novamente.

- CP18: O Gerenciador de Avatares e NPCs é o componente do sistema que irá manter os jogadores e NPCs presentes na região administrada pelo Servidor de Região. O Gerenciador de Avatares e NPCs é responsável por controlar a entrada e saída de avatares e NPCs da reigão. O componente também deverá notificar os avatares e NPCs das mudanças de estado dos objetos dentro de seu campo de visão.

- CP19: O Gerenciador de Habilidades é o componente responsável por validar as habilidades utilizadas por avatares ou NPCs e calcular os resultados do uso das habilidades. Deverá ser considerado informações do lançador da habilidade e do(s) afetado(s) pela habilidade, como chance de bloqueio, aparo, acerto crítico e esquiva.

- CP20: O NPC AI é a parte do sistema que deverá decidir as ações dos NPCs. Utilizando informações vindas do componente Gerenciador de Avatares e NPCs deverá calcular a movimentação e decidir as habilidades que serão utilizadas.

- CP21: O Gerenciador de Regiões é responsável pela organização das diversas instâncias do Servidor de Região. Caso o Servidor de Região estiver administrando uma área com mais mais avatares e NPCs do que suporta, uma nova instância do Servidor de Região deverá ser iniciada, dividindo a região administrada. Caso o Servidor de Região estiver subutilizado, este deverá se comunicar com os Servidores de Regiões Vizinhas para verificar se algum também está subutilizado e, caso positivo, apenas uma instância do Servidor de Região deverá administrar ambas as regiões. O Gerenciador de Regiões deverá manter a lista de vizinhos atualizadas nos Servidores de Região envolvidos na criação ou deleção de uma instância.

- CP22: O Gerenciador de Colisões é a parte do servidor que irá calcular a posição resultante da movimentação dos avatares e NPCs. Obstáculos deverão ser considerados no algoritmo para não permitir que estes objetos ultrapassem objetos rígidos do mundo virtual.

- CP23: O Gerenciador de Combates é o componente do sistema que gerencia os combates entre jogadores e NPCs, jogadores contra jogadores e NPCs contra NPCs. O Gerenciador de Combates utiliza informações vindas do Gerenciador de Habilidades para atualizar os pontos de vida e recurso dos envolvidos no combate. Este módulo também gera os itens resultantes do combate.

- CP24: Os Servidores de Regiões Vizinhas são as instâncias do Servidor de Região que administram as áreas vizinhas à área administrada. Os Servidores de Região deverão se comunicar os com Servidores de Regiões Vizinhas para poder notificar seus avatares e NPCs corretamente, já que o campo de visão de seus avatares e NPCs podem extrapolar a área administrada.

Os conectores são descritos abaixo:

- CN8: A Memória Compartilhada é o conector que permite que diversos componentes trabalhem em uma mesmo conjunto de dados. O conjunto de dados mantidos pelo conector são os dados de posicionamento e pontos de vida e recursos dos avatares e NPCs administrados pelo Servidor de Região. A Memória Compartilhada deverá implementar o modelo arquitetural de *Blackboard* (DEWITT; GRAY, 1992).

- CN9: A Base de Rede Local é a parte do Servidor de Região que irá realizar a comunicação entre as diversas instâncias do Servidor de Região que administram áreas vizinhas. Diferentemente da Base de Rede do servidor e do cliente, este conector irá se comunicar por uma rede local de alto desempenho, logo deverá se comportar de maneira diferente para otimizar o uso da rede.

Os conectores "Gerenciador de Mensagens" e "Gerenciador de Cache" presentes na Fig. (4) são os conectores apresentados na Fig. (3) que se relacionam com os Servidores de Região.

2.4 Resultados da Etapa 4

Conforme apresentado na seção 1.6.4, a Etapa 4 consistiu na definição da parte do framework que será implementado na segunda parte do trabalho.

Segundo Huang, Ye e Cheng (2004) a performance é o aspecto primordial para viabilização de um MMORPG. Por esta razão optou-se por priorizar os requisitos que mais impactam na performance.

Os requisitos com impacto significativo na performance são: RF2, RF3, RF5, RF14, RNF3, RNF4, RNF6, RNF7, RNF8, RNF9, RNF10, RNF11 e RNF12.

Para visualizar o envolvimento dos componentes e conectores com os requisitos funcionais e não funcionais foi feito o mapeamento entre eles nas Tabs. (1) e (2).

Uma importante observação da Tab. (1) é que o requisito RF1 não é atendido e o requisito RF2 é parcialmente atendido pelo framework. Tais responsabilidades são atribuídas à parte web e ao atualizadorr do cliente.

Assim é possível observar que os componentes CP9, CP10, CP11, CP,12, CP18 e os conectores CN2, CN3, CN4, CN5, CN6, CN7 deverão ser implementados na segunda parte do trabalho.

	RF1	RF2	RF3	RF4	RF5	RF6	RF7	RF8	RF9	RF10	RF11	RF12	RF13	RF14	RF15
CP1	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-
CP2	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-
CP3	-	-	-	-	-	-	x	-	-	-	-	-	-	-	-
CP4	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-
CP5	-	-	-	-	-	-	-	-	-	x	-	-	-	-	-
CP6	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-
CP7	-	-	-	-	-	-	-	-	-	x	-	-	-	-	-
CP8	-	-	-	x	-	-	-	-	-	-	-	x	-	-	-
CP9	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-
CP10	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-
CP11	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-
CP12	-	-	x	x	x	-	-	-	-	-	-	x	-	-	-
CP13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x
CP14	-	-	-	-	-	-	-	-	-	-	-	x	-	-	-
CP15	-	-	-	-	-	-	-	-	x	x	-	x	-	-	-
CP16	-	-	-	-	-	-	-	-	-	-	-	x	-	-	-
CP17	-	-	-	-	-	-	-	-	-	-	-	-	x	-	-
CP18	-	-	-	-	x	x	x	-	-	-	-	-	-	-	-
CP19	-	-	-	-	-	-	-	x	-	-	x	-	-	-	-
CP20	-	-	-	-	-	-	-	-	-	-	x	-	-	-	-
CP21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CP22	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-
CP23	-	-	-	-	-	-	-	-	-	-	x	-	-	-	-
CP24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CN1	-	-	-	-	-	-	x	x	-	-	-	-	-	-	-
CN2	-	-	-	-	-	-	x	-	-	-	-	-	-	-	-
CN3	-	-	x	-	-	-	x	x	x	x	-	-	x	-	-
CN4	-	-	x	-	-	-	x	x	x	x	-	-	x	x	-
CN5	-	-	-	-	-	-	x	-	-	-	-	-	-	-	-
CN6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CN7	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-
CN8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CN9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Tabela 1 – Mapeamento dos requisitos funcionais com os componentes e conectores.

2.5 Resultados da Etapa 5

Conforme apresentado na seção 1.6.5, a Etapa 5 consistiu na implementação da parte alvo do framework. Esta etapa compara a arquitetura descritiva apresentada na Etapa 3 com a arquitetura implementada visando a análise da degradação da arquitetura.

O código desenvolvido está disponível em https://github.com/talesms/nix.

Esta seção analisa primeiramente os conectores implementados e em seguida discute os componentes implementados.

	RNF1	RNF2	RNF3	RNF4	RNF5	RNF6	RNF7	RNF8	RNF9	RNF10	RNF11	RNF12
CP1	-	-	-	-	-	-	-	-	-	-	-	-
CP2	-	-	-	-	-	-	-	-	-	-	-	-
CP3	-	-	-	-	-	-	-	-	-	-	-	-
CP4	-	-	-	-	-	-	-	-	-	-	-	-
CP5	-	-	-	-	-	-	-	-	-	-	-	-
CP6	-	-	-	-	-	-	-	-	-	-	-	-
CP7	-	-	-	-	-	-	-	-	-	-	-	-
CP8	-	-	-	-	-	-	-	-	-	-	-	-
CP9	-	-	-	-	-	-	-	-	-	-	-	-
CP10	-	-	-	-	-	-	-	-	-	-	-	-
CP11	-	-	-	-	-	-	-	-	-	x	x	x
CP12	-	-	-	-	-	-	-	-	-	x	x	x
CP13	-	-	-	-	x	-	-	-	-	-	-	-
CP14	-	-	-	-	-	-	-	-	-	-	-	-
CP15	-	-	-	-	-	-	-	-	-	-	-	-
CP16	-	-	-	-	-	-	-	-	-	-	-	-
CP17	-	-	-	-	-	-	-	-	-	-	-	-
CP18	x	-	-	-	-	-	x	x	-	-	-	-
CP19	-	-	-	-	-	-	-	-	-	-	-	-
CP20	-	-	-	-	-	-	-	-	-	-	-	-
CP21	x	x	-	-	-	-	-	-	-	-	-	-
CP22	-	-	-	-	-	-	-	-	-	-	-	-
CP23	-	-	-	-	-	-	-	-	-	-	-	-
CP24	x	x	-	-	-	-	-	-	-	-	-	-
CN1	-	-	-	-	-	-	-	-	-	-	-	-
CN2	-	-	x	-	-	-	-	-	-	-	-	-
CN3	x	x	x	x	-	-	-	x	x	-	-	-
CN4	x	x	x	x	-	x	-	x	x	-	-	-
CN5	-	-	x	-	-	-	-	-	-	-	-	-
CN6	x	x	x	-	-	-	-	-	-	-	-	-
CN7	-	-	-	-	-	-	-	-	-	-	-	-
CN8	x	x	-	-	-	-	-	-	-	-	-	-
CN9	x	x	-	-	-	-	-	-	-	-	-	-

Tabela 2 – Mapeamento dos requisitos não funcionais com os componentes e conectores.

2.5.1 Conectores

Os conectores são de grande importância para a performance do sistema já que realizam transferência de controle e dados entre componentes, gerenciando as mensagens que trafegam na rede e otimizando as chamadas à outros componentes. Os conectores implementados foram:

- Gerenciador de Mensagens do Cliente (CN2)
- Base de Rede do Cliente (CN3)
- Base de Rede do Servidor (CN4)

- Gerenciador de Mensagens do Servidor (CN5)
- Gerenciador de Cache (CN6)
- Gerenciador de Consultas Estruturadas (CN7)

Nas subseções seguintes é discutido cada um destes conectores.

2.5.1.1 Gerenciador de Mensagens do Cliente (CN2)

O Gerenciador de Mensagens do Cliente é responsável por propagar as mensagens recebidas do servidor pelo conector "Base de Rede do Cliente" para os componentes do cliente e propagar as mensagens recebidas dos componentes do cliente para a "Base de Rede do Cliente" enviar para o servidor. Como e apresentado na interface da classe "MessageDelivery":

```
1  #ifndef _NIX_MESSAGEDELIVERY
2  #define _NIX_MESSAGEDELIVERY
3
4  #include "Message.h"
5  #include "Publisher.h"
6  #include "Configuration.h"
7  #include "../Network.h"
8
9
10 class MessageDelivery : public Publisher
11 {
12 public:
13     MessageDelivery(Network* network);
14     ~MessageDelivery();
15
16     void deliverToModule(Message* message);
17     void deliverToServer(Message* message);
18
19 private:
20     Network* network;
21 };
22
23 #endif
```

Listing 2.1 – client/core/MessageDelivery.h

Este conector utiliza o padrão de projeto *Publisher-Subscriber* que ajuda a manter sincronizado o estado de componentes que cooperam entre si. O *Publisher*, este conector,

publica as mensagens para todos os *Subscribers*, os demais módulos do servidor previamente inscritos na lista de assinantes. Durante a inscrição, os *Subscribers* informam o tipo de mensagem desejada. Os destinatários a serem notificados pela mensagem se restringirão aos previamente inscritos naquele tipo. Onde o *Publisher* é definido como:

```
1  #ifndef _NIX_PUBLISHER
2  #define _NIX_PUBLISHER
3
4  #include <vector>
5  #include <iostream>
6
7  #include ``Subscription.h''
8  #include ``Message.h''
9
10 class Publisher
11 {
12 public:
13         Publisher();
14         ~Publisher();
15
16         void subscribe(Subscription* subscription);
17         void unsubscribe(Subscription* subscription);
18         void notifySubscribers(Message* message);
19         void notifySubscribersByOptions(Message* message);
20 private:
21         std::vector<Subscription*> subscriptions;
22 };
23
24 #endif
```

Listing 2.2 – client/core/Publisher.h

Sendo "Subscription" constituído de um *Subscriber* e um tipo de mensagem, os *Subscribers* podem se inscrever (linha 16) e cancelar sua inscrição (linha 17) a qualquer momento durante a execução do sistema, economizando, assim, recursos de rede quando as mensagens não são necessárias.

2.5.1.2 Base de Rede do Cliente (CN3)

A Base de Rede do Cliente é responsável por realizar a comunicação do cliente com o servidor. Este conector é consideravelmente mais simples que a "Base de Rede do Servidor", já que o cliente enxerga o servidor como uma entidade com apenas um endereço IP e uma porta.

Este conector foi implementado utilizando soquete de rede, que são primitivas do serviço de transporte para utilização da camada de rede dos computadores de maneira adequada e fácil de usar. Foi escolhido o protocolo TCP (Transmission Control Protocol) que, além de apresentar boa performance para sistemas de MMORPG segundo Griwodz e Halvorsen (2006), apresenta confiabilidade devido a sua orientação à conexão, por utilizar técnicas para proporcionar uma entrega confiável dos pacotes de dados, permitindo a recuperação de pacotes perdidos, a eliminação de pacotes duplicados, a recuperação de dados corrompidos e ligação em caso de problemas no sistema e na rede (TANENBAUM; WETHERALL, 2010).

Notas aos usuários do framework: Para que a Base de Rede do Cliente funcione corretamente é necessário escrever o endereço IP e a porta da "Base de Rede do Servidor" no arquivo "Configuracao.txt" na pasta "client".

2.5.1.3 Base de Rede do Servidor (CN4)

A Base de Rede do Servidor é o conector mais importante para a performance do sistema, ele otimiza a troca de mensagens entre clientes e servidor. Para encapsular as operações de rede no sistema foi criada a classe "BaseNet", que é também utilizada nos demais componentes que executam em diferentes computadores no lado do servidor.

Para a Base de Rede do Servidor, a classe "BaseNet" é responsável por repassar as mensagens recebidas dos clientes para o "Gerenciador de Mensagens do Servidor" e enviar as mensagens recebidas do "Gerenciador de Mensagens do Servidor" para os clientes ou outros módulos.

Assim como o "Base de Rede do Cliente" este conector foi implementado utilizando soquete de rede pelos motivos de performance e confiabilidade apresentados anteriormente.

Notas aos usuários do framework: Este conector está bem consolidado e não é recomendado a sua alteração. Para que este componente funcione corretamente deve-se escrever os endereços IP, portas e chaves dos módulos do servidor no arquivo "Configuracoes.txt" que se encontra na pasta "server".

2.5.1.4 Gerenciador de Mensagens do Servidor (CN5)

O Gerenciador de Mensagens do Servidor é o conector que dissemina eventos no sistema distribuido, permitindo que os componentes operem independentemente e com baixo acoplamento. Este conector gerencia as mensagens que são trocadas entre cliente e os módulos do servidor.

Assim como o "Gerenciador de Mensagens do Cliente" este conector utiliza o padrão de projeto *Publisher-Subscriber* que ajuda a manter sincronizado o estado de componentes que cooperam entre si. Onde o *Publisher* é este conector e os *Subscribers* são

os demais módulos do servidor. Este conector é mais complexo que o "Gerenciador de Mensagens do Cliente" como é evidenciado no código abaixo:

```
1  #ifndef _NIX_MESSAGEDELIVERY
2  #define _NIX_MESSAGEDELIVERY
3
4  #include "BaseNet.h"
5  #include "Message.h"
6  #include "Publisher.h"
7  #include "LoginMessage.h"
8  #include "CharacterListMessage.h"
9  #include "Configuration.h"
10
11 #define SIZE_AUTORIZATION_KEY 14
12
13 class BaseNet; //Circular Dependency
14
15 class MessageDelivery : public Publisher
16 {
17 public:
18     MessageDelivery(BaseNet* baseNet);
19     ~MessageDelivery();
20
21     void deliverToModule(Message* message);
22     void deliverToClient(Message* message);
23     void deliverToCentral(Message* message);
24     void setCentralRequestListener();
25     LoginMessage* requestCentraltoLoginClient(int clientSock);
26     char requestCentralToSendCharacterList(int clientSock,
          CharacterListMessage* characterListMessage);
27
28 private:
29     BaseNet* baseNet;
30     char* requestHostname;
31     int requestPortno;
32     char* requestKey;
33 };
34
35 #endif
```

Listing 2.3 – server/core/MessageDelivery.h

Este conector apresenta mais opções de entrega de mensagens. Ele permite que os módulos troquem mensagens entre si (linha 23) e que seja feita a o envio de mensagens

em uma nova *thread* (linha 25 e 26) para mensagens que irão gerar uma resposta que possuem pelo menos um comando bloqueante e ou tenha uma Mensagem que ultrapasse o tamanho padrão.

2.5.1.5 Gerenciador de Cache (CN6)

O Gerenciador de Cache é o conector que possibilita aos módulos do servidor acessar as informações persistidas no banco de dados. Para maior performance, são carregadas para a memória RAM os dados das tabelas que só se alteram durante a manutenção do servidor e são requisitados constantemente.

Para acessar o banco de dados MySQL este conector utiliza a biblioteca "mysql++" que é uma dependência do framework que deve ser instalado no sistema operacional.

Este conector utiliza a classe "DataBase" que encapsula o acesso ao banco de dados e a classe "CacheManager" que é a interface para que os componentes do servidor utilizem o banco de dados.

O conector recebe as requisições das informações em cache em uma *thread* diferente da *thread* de requisições de informações estáticas. Assim as requisições das informações estáticas, que são mais demoradas, não atrasam as requisições das informações em cache. Conforme pode ser visto na Fig. (5).

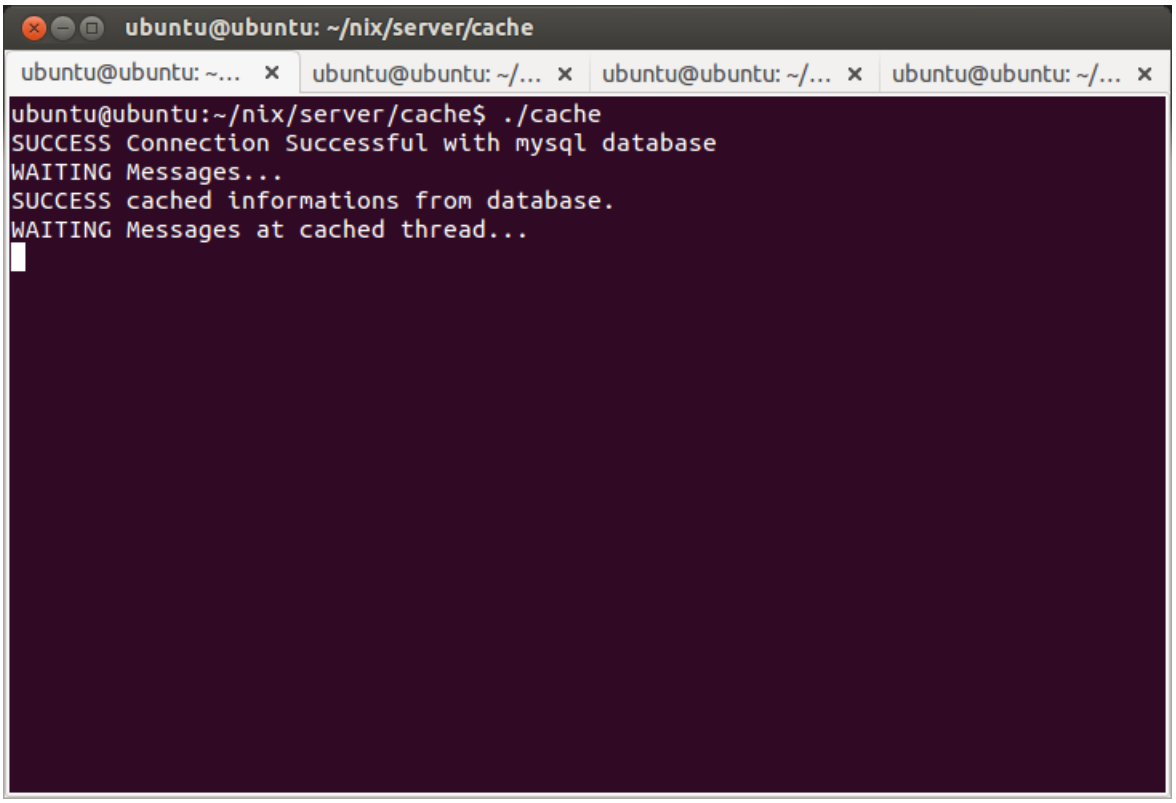

Figura 5 – Execução do Gerenciador de Cache

Notas aos usuários do framework: Para que este conector funcione corretamente deve-se escrever os endereços IPs, portas e chaves no arquivo "Configuracoes.txt" na pasta "server/db".

2.5.1.6 Gerenciador de Consultas Estruturadas (CN7)

O Gerenciador de Consultas Estruturadas foi incorporado ao Gerenciador de Cache para simplificar o sistema, já que o "Banco de Dados de Contas" e o "Banco de Dados do Jogo" se uniram em um mesmo banco de dados. As funcionalidades de consultas estruturadas ao banco de dados agora é feita pelo Gerenciador de Cache, que deixa transparente para o usuário do banco se está ocorrendo a otimização por meio de cache ou não.

A operação de login do usuário é um exemplo de consulta que não é executada cache pelo conector já que estes dados dificilmente serão reutilizados em um tempo próximo.

Notas aos usuários do framework: Deve-se estender o conector para abranger as novas tabelas no banco de dados. O acesso ao banco nunca deverá ser feito diretamente pelos módulos do servidor, pois aumentaria o acoplamento do sistema e diminuiria a coesão.

2.5.2 Componentes

Os componentes caracterizam o sistema como um framework de MMORPGs, já que definem serviços específicos deste domínio. Foram implementados os componentes que possuíam maior impacto na performance e que caracterizavam o sistema como massivamente multijogador, estes componentes foram:

- Servidor de Login (CP9)
- Servidor de Região (CP10)
- Banco de Dados de Contas (CP11)
- Banco de Dados do Jogo (CP12)
- Gerenciador de Avatares e NPCs (CP18)

Nas subseções seguintes é discutido cada um destes componentes.

2.5.2.1 Servidor de Login (CP9)

O Servidor de Login é um servidor conectado à rede interna do provedor de serviço de MMORPG que é responsável por autorizar ou não o acesso de um usuário. Caso seja

um usuário válido este módulo obtém os avatares deste jogador e informa ao gerenciar de avatares e NPCs do servidor região que o novo jogador se conectou.

Como o Servidor de Login pode residir em um computador separado na rede do servidor, o componente utiliza a classe "BaseNet", citada anteriormente, para se registrar como módulo no "Gerenciador de Mensagens do Servidor".

Como é possível observar no trecho de código abaixo, é criado uma thread para realizar o login de cada novo jogador que se conecta no sistema.

```
1  void InitPlayers::update(Message* message)
2  {
3          pthread_t* newThread;
4          int sock;
5          struct ThreadParam* threadParam;
6          threadParam = (struct ThreadParam*) malloc(sizeof(struct
               ThreadParam));
7
8          newThread = new pthread_t();
9          sock = message->getClientSocket();
10
11         threadParam->sock = sock;
12         threadParam->messageDelivery = messageDelivery;
13         threadParam->initPlayers = this;
14
15         std::cout << ``SUCCESS a player has connected, sock: ''
               <<  sock << std::endl;
16
17         pthread_create(newThread, NULL, newClientThreadFunction,
               (void*) threadParam);
18 }
```

Listing 2.4 – server/login/InitPlayers.cpp função update

Lançando uma nova thread para cada requisição de login permite o sistema atender vários usuários simultaneamente, sem que o atraso de conexão de alguns jogadores interfiram no tempo de resposta do servidor para os demais jogadores. A comunicação com o cliente se dá por meio do Gerenciador de Mensagens do Servidor (linha 12).

2.5.2.2 Servidor de Região (CP10)

O servidor de região é um grande componente onde são realizados todos os acontecimentos de uma parte do mundo virtual. Podendo existir várias instâncias deste servidor, de modo a abranger toda a área do mundo virtual. Este componente engloba o componente implementado Gerenciador de Avatares e NPCs.

Diferentemente dos outros módulos do servidor, o Servidor de região é um módulo que possui módulos internos devido a sua complexidade. Para o baixo acoplamento e alta coesão destes módulos internos utilizou-se também o padrão de projeto *Publisher-Subscriber*, assim todas as mensagens recebidas pelo Servidor de Região podem ser repassadas corretamente para seus módulos internos que se inscreveram para aquele tipo de mensagem.

Notas aos usuários do framework: Caso seja desejável incluir mais módulos internos no Servidor de Região deve-se inscrevê-los no gerenciador de mensagens deste componente. Deve-se também escrever os endereços IP, portas e chaves no arquivo "Configuracoes.txt" na pasta "server/region" para que o Servidor de Região disponibilize seu serviço na rede.

2.5.2.3 Banco de Dados de Contas (CP11)

O sistema possui as contas de usuários persistidas em um banco de dados MySQL. Neste banco foi criado uma tabela "user" onde cara linha representa um usuário do sistema, as colunas "username" e "password" são utilizadas no servidor de login para autenticar o usuário.

Conforme apontado por Chen (2007) o banco MySQL é o banco de dados com a melhor performance para MMORPGs e este também é utilizado por sete dos dez maiores MMORPGs em 2007.

As tabelas do banco de dados podem ser visualizadas na Fig. (6).

Notas aos usuários do framework: Deve-se instalar em uma máquina na rede interna do servidor o serviço de banco de dados do MySQL e executar os comandos SQL contidos na pasta "server/db/sql" para gerar as tabelas necessárias para que o framework funcione corretamente.

2.5.2.4 Banco de Dados do Jogo (CP12)

Para permitir o balanceamento dos itens e habilidades no jogo foi criado tabelas para itens e habilidades no banco de dados MySQL. Estes possuem modificadores para os atributos dos avatares, como pontos de vida e força. Por motivo de simplificação do framework, este banco foi criado juntamente com o "Banco de Dados de Contas", esta decisão não impacta a arquitetura negativamente pois ambos os bancos passam a ter um único ponto de acesso, o conector "Gerenciador de Cache", que de modo transparente otimiza a utilização destes bancos.

Como este banco foi unificado com o "Banco de Dados de Contas", sua configuração e decisões de implementação seguem o que foi apresentado na subseção anterior.

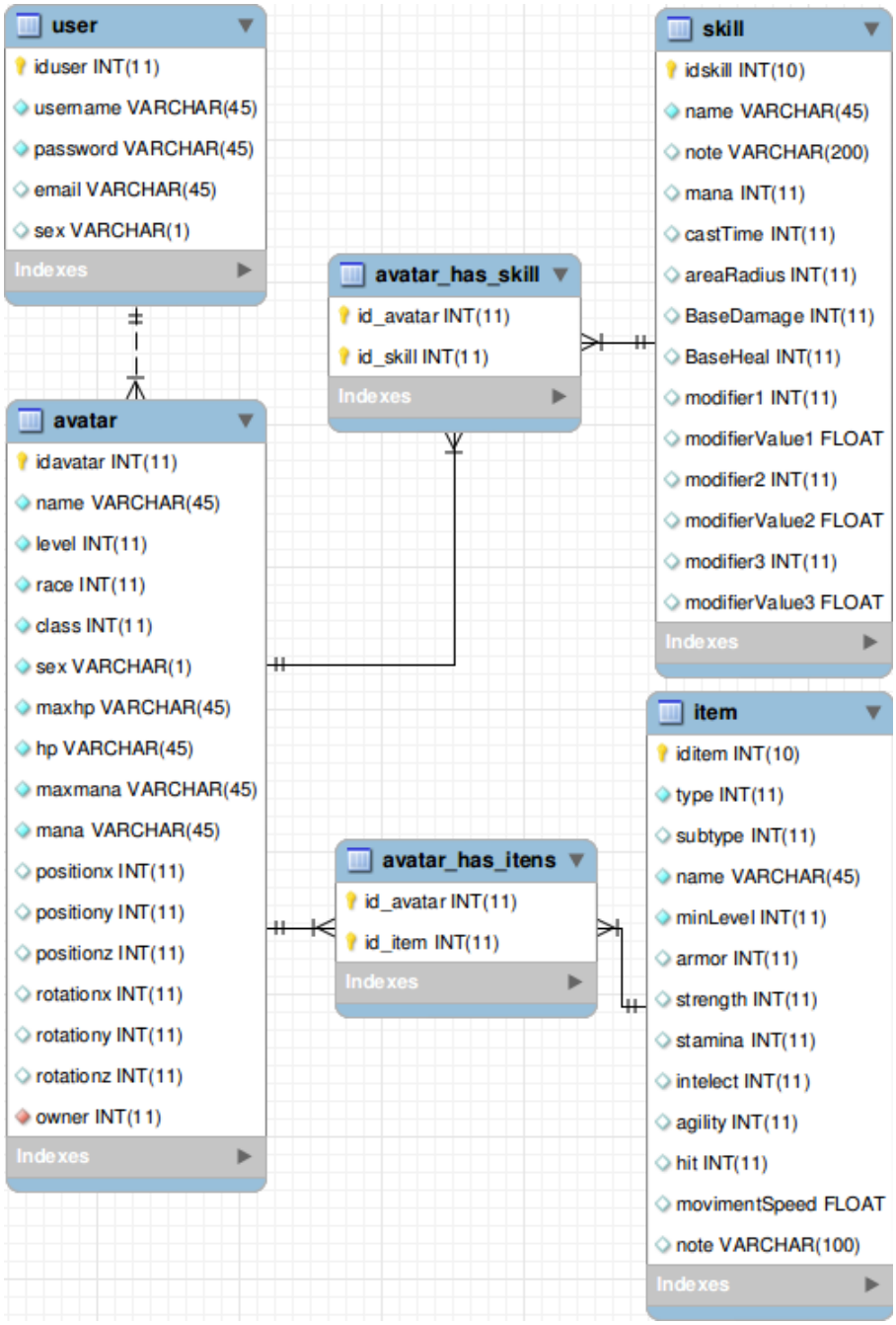

Figura 6 – Tabelas do banco de dados

2.5.2.5 Gerenciador de Avatares e NPCs (CP18)

O componente Gerenciador de Avatares e NPCs executa uma thread diferente para cada avatar, como é possível observar no trecho de código abaixo, para que os atrasos das conexões dos jogares não prejudique os jogares que estão com conexões rápidas. Este é um dos componentes onde o utilizador do framework mais irá estender para atribuir as características específicas de seu MMORPG.

```
1 void AvatarNpcManager::newConnection(Message* message)
2 {
```

```
3     pthread_t* newThread;
4     int sock;
5     struct ThreadParam* threadParam;
6     threadParam = (struct ThreadParam*) malloc(sizeof(struct
         ThreadParam));
7
8     newThread = new pthread_t();
9     sock = message->getClientSocket();
10
11    connections[sock] = newThread;
12
13    threadParam->sock = sock;
14    threadParam->messageDelivery = messageDelivery;
15    threadParam->cacheSock = connectToCacheServer();
16    threadParam->avatarId = message->getValue();
17
18    std::cout << "SUCCESS a player has connected, sock: "<<  sock
          << std::endl;
19
20    pthread_create(newThread, NULL, clientThreadFunction, (void*)
          threadParam);
21 }
```

Listing 2.5 – server/region/AvatarNpcManager.cpp função newConnection

A *thread* iniciada na linha 20 da função "newConnection" executa a função "clientThreadFunction" apresentada no código abaixo.

```
1  Message* customAvatarTick(Avatar* avatar)
2  {
3      //Your implementation goes here
4
5      return NULL;
6  }
7
8  void *clientThreadFunction(void* threadParam)
9  {
10     int sock = ((struct ThreadParam*)threadParam)->sock;
11     int avatarId = ((struct ThreadParam*)threadParam)->avatarId;
12     Message* message;
13     MessageDelivery* messageDelivery = ((struct ThreadParam*)
          threadParam)->messageDelivery;
14     int cacheSock = ((struct ThreadParam*)threadParam)->cacheSock
          ;
```

```
15      Avatar* avatar;
16
17      delete ((struct ThreadParam*) threadParam);
18
19      avatar = loadAvatarFromDatabase(sock, avatarId, cacheSock);
20
21      if(avatar != NULL)
22          std::cout << "SUCCESS the "<< avatar->name << " was
                loaded successfully." << std::endl;
23      else
24          std::cout << "ERROR avatar not found in database." << std
                ::endl;
25
26      while(true)
27      {
28          message = customAvatarTick(avatar);
29
30          if(message != NULL)
31              messageDelivery->deliverToCentral(message);
32      }
33  }
```

Listing 2.6 – server/region/AvatarNpcManager.cpp funções customAvatarTick e clientThreadFunction

O avatar é carregado do banco de dados (linha 19) e se inicia o loop que irá gerenciar os eventos deste avatar (linha 26).

Notas aos usuários do framework: O comportamento esperado de cada avatar irá variar de jogo para jogo, logo é necessário que estes comportamentos sejam implementados na função "customAvatarTick" apresentada na linha 1. Esta função permite a alteração dos dados do Avatar por meio do ponteiro recebido de parâmetro e permite que sejam enviadas mensagens para outros módulos ou para o cliente gerenciado por aquela *thread* por meio do retorno da função.

3 Considerações Finais

Este trabalho abordou o gênero MMORPG de jogos que possui uma importante e crescente parcela no mercado de jogos. Entretanto, neste gênero observa-se uma lacuna ferramental que favoreça o reúso durante a sua implementação.

Com o intuito de atenuar esta lacuna ferramental foi proposto um framework com desenvolvimento orientado pela arquitetura. Arquitetura desenvolvida com base em estudos bibliográficos, análise do MMORPG que possuía 62,4% do mercado em 2008, World of Warcraft, e análise do MMORPG de código livre Planeshift.

Este trabalho apresenta as seguintes contribuições para os desenvolvedores de sistemas desta natureza: delimitação e reconhecimento de domínio de MMORPGs, definição dos requisitos que definem um jogo como MMORPG, representação arquitetural de um framework para MMORPG e implementação da parte do framework proposto que mais impacta a performance.

A delimitação e reconhecimento do domínio de MMORPGs é de grande interesse de todos os desenvolvedores de MMORPG, pois este estudo identifica e caracteriza os principais problemas técnicos encontrados no desenvolvimento deste tipo de sistema. Este reconhecimento de domínio também apresenta as soluções adotadas para estes problemas.

A definição dos requisitos funcionais e não funcionais auxiliará os desenvolvedores de MMORPG a entender os objetivos do sistema de modo a atender as expectativas dos jogadores deste gênero.

A representação arquitetural permitirá que os desenvolvedores reutilizem várias decisões de projeto de forma a atender os requisitos do sistema. Aumentando assim atributos de qualidade de software como baixo acoplamento e alta coesão.

A implementação da parte alvo do framework busca garantir a performance, principal aspecto para a viabilização destes sistemas. Espera-se que a reutilização desta implementação garanta a performance desejada para sistemas MMORPG.

O trabalho de conclusão de curso observou aspectos técnicos relativos ao desenvolvimento do framework que o preparam para oferecer alto desempenho, entretanto não houve uma avaliação quantitativa do sobre o ganho de desempenho que as decisões de arquitetura provocaram. Essa perspectiva pode ser um trabalho futuro muito pertinente. Além disso pode-se implementar o restante do framework para difundir seu uso, validar o framework com os requisitos definidos para verificar se está atingindo seus objetivos, e criar testes automatizados para redução de erros.

Referências

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. Addison Wesley Professional, 2003. (The SEI Series in Software Engineering). ISBN 9780321154958. Disponível em: <http://books.google.com.br-/books?id=mdiIu8Kk1WMC>. Citado 3 vezes nas páginas 18, 21 e 22.

BUSCHMANN, F. et al. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Chichester, UK: Wiley, 1996. ISBN 978-0-471-95869-7. Citado 2 vezes nas páginas 35 e 37.

CHEN, K.-T. et al. Game traffic analysis: an mmorpg perspective. In: *Proceedings of the international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2005. (NOSSDAV '05), p. 19–24. ISBN 1-58113-987-X. Disponível em: <http://doi.acm.org/10.1145/1065983.1065988>. Citado na página 17.

CHEN, K. ta et al. Identifying mmorpg bots: A traffic analysis approach. In: *ACE2006 (Los Angeles 14 th - 16 th*. [S.l.]: SpringerVerlag, 2006. Citado 2 vezes nas páginas 28 e 37.

CHEN, W. *Open software to innovation: the critical success factors of massively multiplayer online role playing games (MMORPG) in China and Ireland*. Tese (Doutorado) — Dublin Institute of Technology, Zürich, Switzerland, 2007. Citado na página 50.

CORNETT, S. The usability of massively multiplayer online roleplaying games: designing for new users. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2004. (CHI '04), p. 703–710. ISBN 1-58113-702-8. Disponível em: <http://doi.acm.org/10.1145/985692.985781>. Citado na página 17.

DEWITT, D.; GRAY, J. Parallel database systems: the future of high performance database systems. *Commun. ACM*, ACM, New York, NY, USA, v. 35, n. 6, p. 85–98, jun. 1992. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/129888.129894>. Citado 2 vezes nas páginas 37 e 40.

DIOT, C.; GAUTIER, L. A distributed architecture for multiplayer interactive applications on the internet. *Network, IEEE*, v. 13, n. 4, p. 6–15, 1999. ISSN 0890-8044. Citado 3 vezes nas páginas 25, 30 e 31.

FAYAD, M.; SCHMIDT, D. C. Object-oriented application frameworks. *Commun. ACM*, ACM, New York, NY, USA, v. 40, n. 10, p. 32–38, out. 1997. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/262793.262798>. Citado 3 vezes nas páginas 19, 20 e 21.

GAMMA, E. et al. *Design patterns: elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2. Citado na página 18.

GRIWODZ, C.; HALVORSEN, P. The fun of using tcp for an mmorpg. In: *Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2006. (NOSSDAV '06, v. 1), p. 1:1–1:7. ISBN 1-59593-285-2. Disponível em: <http://doi.acm.org/10.1145/1378191.1378193>. Citado 4 vezes nas páginas 26, 27, 36 e 45.

HENTTONEN MATINLASSI, N.; KANSTéN. Integrability and extensibility evaluation from software architectural models - a case study. *Open Software Engineering Journal*, v. 1, p. 1–20, 2007. Disponível em: <http://www.bentham-open.org/pages-/content.php?TOSEJ/2007/00000001/00000001/1TOSEJ.SGM>. Citado na página 20.

HUANG, G.; YE, M.; CHENG, L. Modeling system performance in mmorpg. In: *Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE*. [S.l.: s.n.], 2004. p. 512–518. Citado 2 vezes nas páginas 17 e 40.

ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. [S.l.]: ISO/IEC, 2001. Citado na página 20.

KAWALE, J.; PAL, A.; SRIVASTAVA, J. Churn prediction in mmorpgs: A social influence based approach. In: *Computational Science and Engineering, 2009. CSE '09. International Conference on*. [S.l.: s.n.], 2009. v. 4, p. 423–428. Citado na página 17.

KHARE, R. et al. xADL: enabling architecture-centric tool integration with XML. In: *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*. [s.n.], 2001. p. 9 pp.+. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all-.jsp?arnumber=927248>. Citado na página 22.

KNUTSSON, B. et al. Peer-to-peer support for massively multiplayer games. In: *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. [S.l.: s.n.], 2004. v. 1, p. –107. ISSN 0743-166X. Citado 3 vezes nas páginas 18, 26 e 33.

LEE, Y.-T.; CHEN, K.-T. Is server consolidation beneficial to mmorpg? a case study of world of warcraft. In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. [S.l.: s.n.], 2010. p. 435–442. Citado 2 vezes nas páginas 19 e 26.

PRESSMAN, R. *Engenharia de software*. McGraw-Hill, 2006. ISBN 9788586804571. Disponível em: <http://books.google.com.br/books?id=MNM6AgAACAAJ>. Citado 2 vezes nas páginas 18 e 20.

RIEHLE, D. *Framework Design: A Role Modeling Approach*. Tese (Doutorado) — ETH Zürich, Zürich, Switzerland, 2000. Citado 2 vezes nas páginas 19 e 20.

SOMMERVILLE, I. et al. *Engenharia de software*. ADDISON WESLEY BRA, 2008. ISBN 9788588639287. Disponível em: <http://books.google.com.br-/books?id=ifIYOgAACAAJ>. Citado 4 vezes nas páginas 20, 22, 29 e 30.

SUZNJEVIC, M.; STUPAR, I.; MATIJASEVIC, M. Traffic modeling of player action categories in a mmorpg. In: *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering),

2011. (SIMUTools '11), p. 280–287. ISBN 978-1-936968-00-8. Disponível em: <http://dl.acm.org/citation.cfm?id=2151054.2151105>. Citado 2 vezes nas páginas 17 e 27.

TANENBAUM, A.; WETHERALL, D. *Computer Networks*. Pearson Prentice Hall, 2010. ISBN 9780132126953. Disponível em: <http://books.google.com.br/books?id=I764bwAACAAJ>. Citado na página 45.

TAYLOR, R. N.; MEDVIDOVIC, N.; DASHOFY, E. M. *Software Architecture: Foundations, Theory, and Practice*. [S.l.]: Wiley Publishing, 2009. ISBN 0470167742, 9780470167748. Citado na página 22.

THAWONMAS, R.; KASHIFUJI, Y.; CHEN, K.-T. Detection of MMORPG bots based on behavior analysis. In: *Proceedings of ACM ACE 2008*. [S.l.: s.n.], 2008. Citado 2 vezes nas páginas 28 e 37.

WOODCOCKL, B. *An Analysis of MMOG Subscription Growth*. [S.l.], 2008. Disponível em: <http:/www.mmogchart.com/>. Citado na página 17.

YAMAMOTO, S. et al. A distributed event delivery method with load balancing for mmorpg. In: *NETGAMES*. ACM, 2005. p. 1–8. ISBN 1-59593-156-2. Disponível em: <http://dblp.uni-trier.de/db/conf/netgames/netgames2005.html>. Citado na página 18.